The Design, Development and Testing of a Multi-process Real-time Software System

Marlin Gendron and Maura Lohrenz
Naval Research Laboratory, Code 7440.1, Stennis Space Center, Mississippi

Abstract

Real-time data acquisition systems that ingest and output data at very high rates require computer software that is written to be efficient and robust. This paper discusses the design and development of a collection of separate software applications that work together to gather, analyze, process, and display high-ping-rate sonar data.

Introduction

Booch (1983), one of the leading pioneers in software engineering, states that developing and completing an efficient, reliable and understandable software system can be a Herculean task, especially when the system is large and operates in real-time. Booch also says the creators of such systems must act as part scientists, part artists, and must develop smaller individual software components separately to realize the larger system.

MacLennan (1987), from the Naval Postgraduate School, agrees with Booch when he states that much of the difficulty of programming large systems stems from the complexity of dealing with many different details at one time. A sound engineering approach is to break the whole system down into smaller manageable parts and develop algorithms and data structures to solve the parts separately.

Over the past few years, the volume of data produced by high-resolution sonars has increased dramatically, and processing of the sonar data has become more complex (Reed, 2001; Cronin et al., 2003). The Naval Research Laboratory (NRL) has developed complex software systems, using the separate-design approach described above, that are capable of acquiring, processing, and displaying large amounts of data as it is being collected, i.e., in real-time.

This paper describes the design, development, and testing of one such system that collects, processes, and displays data from multiple high-ping-rate sonars simultaneously. This system includes NRL-developed automated algorithms capable of aiding sonar analysts to more efficiently perform complex, time-consuming tasks, by presenting helpful visual cues to the analysts as the sonar data is being displayed. Sonar analysts are aided in such tasks as 1) clutter detection, 2) bottom roughness estimation, 3) object detection and 4) change detection.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
MAR 2007		**00-00-2007 to 00-00-2007**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
The Design, Development and Testing of a Multi-process Real-time Software System	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Research Laboratory,Code 7440.1,Stennis Space Center ,MS,39529**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Proceedings of the Industry Engineering and Management Systems (IEMS) 2007 Conference, 12-14 Mar, Cocoa Beach, FL

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**7**	

System Design

Before software development started, NRL identified the most important requirements of the system:

1) All processing and displaying of sonar data must be successfully accomplished on one computer.
2) Data must be simultaneously acquired from multiple sonars at a high ping rate.
3) Preprocessing tasks (e.g., deconvolution, beam forming, and bottom detection) must be performed on the acquired sonar data as it is being collected.
4) Processed sonar data must be displayed to analysts in real-time.
5) Automated processes such as Computer-aided Detection (CAD), Computer-aided Classification (CAC), and Change Detection algorithms must be executed in real-time (i.e., while sonar data is being displayed).
6) Target information must be displayed in two-dimensions (2-D) and, if possible, three-dimensions (3-D).
7) Sonar and target locations must be tracked and plotted on a separate display.

Because the entire system, with all its functionality, must execute on a single computer while acquiring data from multiple sonars operating at a high ping rate, NRL chose an Intel-based computer that houses multiple (at least four) Central Processor Units (CPUs). The Linux operating system (OS), based on UNIX, was chosen because it supports Symmetric Multiprocessor (SMP) operations, allowing computing tasks to be evenly distributed over all CPUs, thus maximizing the entire throughput of the system.

Examining the initial requirements, NRL determined that the system would need at least three separate Graphical User Interfaces (GUIs). The GUI language Qt, from the company Trolltech, was chosen because it 1) supports the programming languages C and C++, in which all the preprocessing and automated algorithms are written, 2) provides many user-friendly GUI controls and 3) is portable to other OS platforms such as Microsoft Windows.

Next, to reduce the complexity of the programming tasks, the system was divided into separate executable processes, which are individually controllable (Deitel, 2004). Memory space for storing data is normally local to the process that created it, and one process cannot naturally communicate with another. There are several specialized methods of Interprocessor Communication (IPC), in which separate processes can access common data and communicate with each other. NRL investigated these methods to determine the best data-sharing and communication methods to meet the seven requirements listed above.

UNIX allows common data to be accessed by all processes using methods such as Message Passing (MP), UNIX Pipe, File Input/Output (IO), and IPC shared memory. In MP, each process has its own private memory, and the processes are connected, usually with a communication protocol called sockets. To send data from one process to another, the sending process must construct a message, reformat the data, and send the message

packet to the other processes via sockets. MP is often slow and involves a lot of overhead (Stevens, 1999).

A UNIX Pipe is a unidirectional data flow between two processes. Forming pipes can be a complex operation and result in data fragmentation or the complete loss of data (Stevens, 1999).

Passing data via File IO is simple. One process writes the data to some device, such as a hard drive, and another process retrieves the data. The target process must somehow know to access that device and read the data. This requires the target process to poll the device (periodically checking to see if data values have changed) or communicate with the sending process in some way. Reading and writing operations to and from a device is often slow.

For actual data movement, IPC shared memory beats everything hands down, because no copying or reformatting of the data is required in most cases. An exception to this is addressed later in this paper. Because shared memory is the fastest form of IPC available (Stevens, 1999), NRL chose it as the primary means for sharing data among all processes within the system. UNIX provides IPC functions to create, control, and destroy shared-memory segments, identified by unique key values and memory addresses.

Using shared memory, separate processes still must somehow know whether or not common data has changed and, if so, what action to take upon the changed data. Data polling, mentioned above, can greatly slow down processes depending on how often the process has to check to see if common data was modified. A good method for one process to quickly notify another process of a change is to use UNIX signals. When a signal is sent from one process to another, a communication record for each process (stored in shared memory) contains a field telling the receiving process what action to take. Figure 1 shows the chosen architecture for the system.

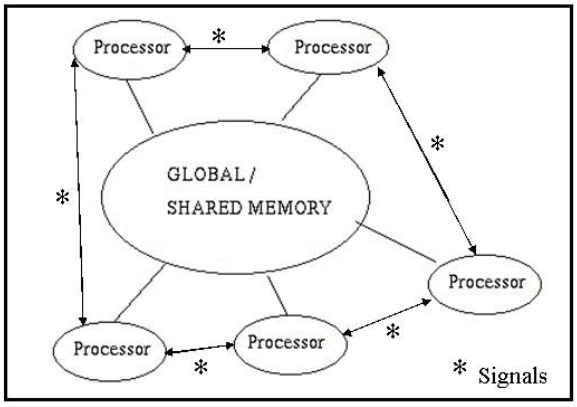

Figure 1. Chosen architecture for real-time data-processing and display system.

Allowing each process to create and destroy shared memory complicates the overall system and can contribute to data corruption. To alleviate this problem, the NRL system is comprised of one primary process, called the Process Manager (PM), which creates

and destroys all shared memory used by all other (secondary) processes. The PM also executes and terminates the secondary processes.

For this system, the PM controls five secondary processes (figure 2): 1) an IO Process responsible for data acquisition and preprocessing, 2) a Display (DISP) process responsible for displaying the sonar data, 3) a Navigation (NAV) process that shows the sonar locations related to targets, 4) a Computer-aided Detection (CAD) process that automatically detects targets and 5) a Computer-aided Classification (CAC) process that automatically classifies the targets.

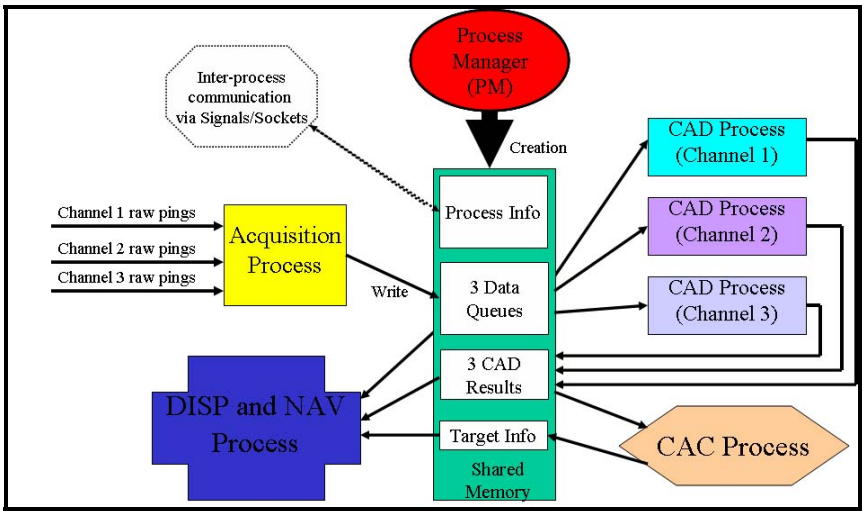

Figure 2. System diagram of the NRL real-time sonar data processing system.

Shared Memory

The PM process and all secondary processes maintain separate link-lists to simplify the tracking and destruction of all shared memory variables. These link-lists contain the following fields: 1) shared-memory key, 2) shared-memory identifier, 3) address to the shared memory, 4) data size of the shared memory, and 5) data length of the shared memory.

Two functions are provided to create and add nodes to each link-list. Because each process has its own link-list, operations performed on each does not affect the link-lists of other processes. NRL provides several other link-list operations that contain IPC shared-memory function calls, which can affect other processes. Because all shared-memory variables have a unique key determined at compile time, and all processes are aware of these keys, functions that destroy, create, and use shared memory can operate directly on memory shared by all the processes.

NRL decided to utilize the One Producer/Many Consumer (OP/MC) data model. Only one process, the producer, is allowed to write to a shared-memory location. The producer sets a write-lock while writing to shared memory to prevent a consumer process from

reading the data at that location before it is ready to be read, thus preventing data corruption. Consumers set a read-lock at the read location preventing other consumers from reading at the same time (figure 3).

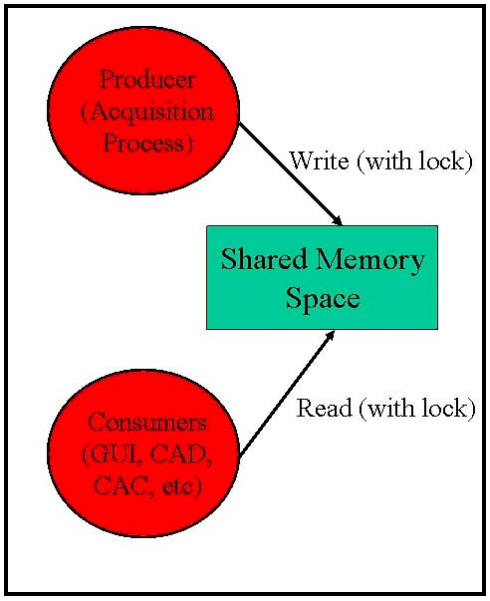

Figure 3. OP/MC model interfaced with shared memory utilizing read and write locks to prevent data corruption.

If the consumer is busy reading shared memory, the producer will have to wait until the consumer is finished before more data can be written to that location. To alleviate this problem, NRL designed circular shared-memory queues (figure 4) allowing the producer to continuously write new data to the queue as long as the queue length is sufficient, based on the reading speed of the consumers.

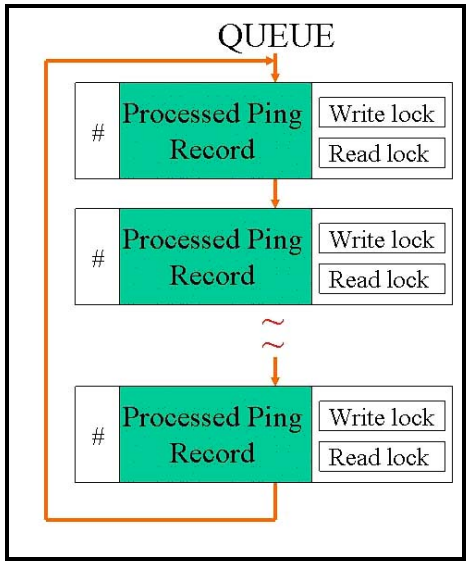

Figure 4. Depiction of a circular data queue that holds processed sonar data.

The producer writes to the next available position in the queue. After the producer writes to the last position in the queue, the next write will occur at the beginning of the queue. There are several issues to consider:

1) If the producer is writing data to the queue faster than a consumer can read it, then the consumer will lose data.
2) The queue can be made longer to help alleviate the problem, but as the queue gets longer, the consumer will fall further behind, causing data lag.
3) There is an upper limit to the size of a queue, determined by the computer resources and OS.

The queues contain a unique record count, number of consumers using the queue, producer queue position, queue size, and a read/write lock array. Each location containing data within the queue is given a unique record number. The next available record is determined from this record count. After a record number is assigned, the record count is incremented by one.

The last field in all the queues is the data record. This system uses queues to store Global Position System (GPS) data, processed data, and target data. The data record for the GPS queue might contain fields such as latitude (LAT), longitude (LON), heading, and time.

In addition, each consumer is allocated a record that tracks their current position in a queue along with the current record number. The length of each queue is important and is normally determined when tuning the system for maximum throughput. Typically, the length of the processed data queue (which is updated many times per second) is around 40 data records, whereas the length of the GPS queue (which is only updated once per second) is only of length 2.

As mentioned earlier, IPC shared memory is fast because no data copying is necessary for most operations. One exception is the case in which one process has to modify shared data but not affect how it is used by other processes. NRL utilizes special functions, called copy functions, which allow producers and consumers to specify which fields in a queue data record will be written to (or read from) and which to ignore. Below is an example of a copy function that reads from the GPS queue. The consumer calling this function only needs LAT and LON values from the GPS queue data record and chooses to ignore other fields, such as heading.

```
GPS *gps_copy (GPS *dest, GPS *src)
{
        dest->lat = src->lat;
        dest->lon = src->lon;
/*
        float   heading;
*/
        return (dest);
}
```

Figure 5. GPS copy function in which only LAT and LON are copied and heading is ignored.

Copy functions allow producers and consumers to write and read faster because not all fields are needed in every situation. Note the field(s) between the C comment markers (/* and */) are not copied in this function (figure 5).

Testing

After all the processes were developed, except the 3-D display, NRL assembled the components and developed the PM, shared memory queues, and signal communication. A sonar data simulator was built to test whether or not the complete system could function on a single computer with multiple sonars outputting data at very high ping rates.

The simulator allowed NRL to conduct software-performance tests without actually having to interface with expensive sonars or conduct the tests in a water environment. The simulator sends a ping of data then waits before sending the next. The wait time can be adjusted to increase or decrease ping rate.

NRL decided that the initial test would involve interfacing the system with multiple simulators, running simultaneously, to measure the maximum ping rate the system could handle without experiencing data lag or loss. A test would be considered successful if the system could remain stable with all simulators outputting data simultaneously at a pre-determined ping rate.

The test was conducted, and the simulators were increased to more than the required ping rate. The system remained stable and was able to preprocess all sonar data, display the data, and perform CAD and CAC operations at the high data rate. Due to the successful test, NRL is now developing the 3-D display process and in the next few months will incorporate that process into the system and repeat the performance test. If the test is unsuccessful, the 3-D process will not be included in the final version of the system.

Conclusion

Real-time data acquisition systems that ingest and output data at very high rates require efficient and robust computer software. Developing smaller individual software components separately to realize a larger system is a common and effective method. NRL developed and tested a complex software system using this approach, capable of acquiring, processing and displaying data from multiple sonars at a high data rate.

A simulator was built to output previously-collected sonar data at the required data rate. NRL conducted performance tests that demonstrate that the current software implementation was sufficient to meet all design criteria. Because not all secondary processes have been fully implemented, especially the 3-D display, further development and testing is needed to determine how these modifications will affect overall system performance.

Acknowledgements

This work was sponsored under Program Element 602435N by the NRL 6.2 Base Program.

References

Booch, G, 1983. *Software Engineering with Ada*. Menlo Park: Benjamin/Coummings Publishing Company.

Cronin, D., M. Broadus, B. Reed, S. Byrne, W. Simmons, and L. Gee, 2003. Hydrographic Work Flow – From Planning to Products. Proceedings of the *U.S. Hydro 2003 Conference*, March 24-27, Biloxi, Mississippi.

Deitel, H.M, P. Deitelaul, and D. Choffnes, 2004. *Operating Systems*. Upper Saddle River, NJ: Pearson/Prentice Hall, ISBN 0-13-182827-4.

MacLennan, B., 1987. *Principles of Programming Languages*. Second Edition, Austin: Holt Rinehart and Winston.

Reed, B., 2001. Data, Data Everywhere and Nary a Bit to Drop. Abstract for the *Shallow Survey 2001 Conference*, Sidney, Australia.

Stevens, R.W., 1999. *UNIX Network Programming, Volume 2, Second Edition: Interprocess Communications*, Prentice Hall, ISBN 0-13-081081-9.